Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в глубину»

Выполнили:

студенты группы 20ВВ2

Киреев Б.П.

Верховский М.В.

Лукин В.Д.

Приняли:

д.т.н. Митрохин М.А.

к.т.н. доцент Юрова О.В.

Пенза 2021

Название: Обход графа в глубину.

Цель работы: Научиться осуществлять обход графа в глубину.

Лабораторное задание:

**Задание 1:**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа G. Выведите матрицу на экран.

2. Для сгенерированного графа осуществите процедуру обхода в глубину, реализованную в соответствии с приведенным выше описанием.

3.\* Реализуйте процедуру обхода в глубину для графа, представленного

списками смежности.

**Задание 2**\***:**

1. Для матричной формы представления графов выполните

преобразование рекурсивной реализации обхода графа к не рекурсивной.

**Задание 1:**

Листинг:

Файл Lab4.cpp

#include <iostream>

#include <ctime>

#include <malloc.h>

#include <conio.h>

using namespace std;

int\* vis = NULL, n, \*spis = NULL, \*\* p = NULL;

int pogr = 0, numbo = 0;

bool check = false, blocker = false; //Переменная для контроля не рекурсивной реализации

//Вершина / След.элемент; Номер

struct SmegnElem {

SmegnElem\* Nextelem;

int num;

};

//Список смежности

struct SmegnList {

SmegnElem\* First;

int Colo;

} \*\*SpecMatrix = NULL, \* SpecMatrixRezerve = NULL;

//Создание списка смежности

SmegnList\* CreateList() {

SmegnList\* groups = (SmegnList\*)malloc(sizeof(SmegnList));

groups->First = NULL;

groups->Colo = 0;

return groups;

}

//Добавление элемента в список смежности

void AddSmegElem(SmegnList\* group, int Chis) {

SmegnElem\* newItem = (SmegnElem\*)malloc(sizeof(SmegnElem));

newItem->Nextelem = NULL;

newItem->num = Chis;

if (group->Colo == 0) { //При отсутствии элементов записываем как первый

group->First = newItem;

group->Colo++;

return;

}

SmegnElem\* last = group->First; //При наличии других элементов начинаем перебор

while (last->Nextelem != NULL) {

last = last->Nextelem;

}

last->Nextelem = newItem;

group->Colo++;

}

//Отрисовка списка смежности

void PrintSmegElem(SmegnList\* groups) {

if (groups->Colo == 0) { //При размере = 0, у нас нет данных

cout << "Список пуст" << endl;

return;

}

SmegnElem\* current = groups->First;

while (current != NULL) { //Перебор структуры до последнего элемента(с выводом)

int value = current->num;

cout << value << " ";

current = current->Nextelem;

}

cout << endl;

}

//Рекурсивный поиск в глубину

void DFS(int num) {

cout << num + 1 << " "; // вывод вершин

vis[num] = 1;//вершина пройдена

for (int i = 0; i < n; i++) {

if (p[num][i] == 1 && vis[i] == 0) { // проверка на посещение ранее

DFS(i);

}

}

}

//Поиск в глубину по спискам смежности

void DeepSearchSmeg() {

for (int i = 0; i < n; i++) { vis[i] = 0; } //Обнуление проходимых вершин

SmegnElem\* Dero = SpecMatrix[0]->First;

int Counter = 1, Numer = 0, Saver = 1;

bool Ruler = false, Tain = false;

vis[Counter - 1] = Counter;

AddSmegElem(SpecMatrixRezerve, 1);

Counter++;

while (Counter != n + 1) {

if (Saver == Numer) {

for (int i = 0; i < n + 1; i++) {

Tain = true;

for (int j = 0; j < n + 1; j++) {

if (vis[j] == i) { Tain = false; };

}

if (Tain == true) { Numer = i - 1; break; }

}

}

Dero = SpecMatrix[Numer]->First;

Tain = false;

Saver = Numer;

while (Tain == false) {

Ruler = false;

for (int i = 0; i < n; i++) { if (Dero->num == vis[i]) { Ruler = true; } }

if (Ruler == false) {

AddSmegElem(SpecMatrixRezerve, Dero->num);

vis[Counter - 1] = Dero->num;

Counter++;

Numer = Dero->num - 1;

Tain = true;

}

if (Dero->Nextelem != NULL) { Dero = Dero->Nextelem; }

else { break; }

}

}

PrintSmegElem(SpecMatrixRezerve);

}

//Не рекурсивный(циклический) поиск в глубину

void NoRecSearch() {

for (int i = 0; i < n; i++) { vis[i] = 0; } //Обнуление проходимых вершин

cout << endl;

while (check != true) {

vis[numbo] = 1;

cout << numbo + 1 << " ";

for (int i = numbo; i < n; i++) {

for (int m = 0; m < n; m++) {

if ((p[i][m] == 1) && (vis[m] == 0)) { vis[m] = 1; i = m - 1; cout << m + 1 << " "; break; }

}

}

check = true;

blocker = false;

for (int i = 0; i < n; i++) {

if (vis[i] == 0) {

check = false;

if (blocker == false) { numbo = i; blocker = true; }

}

}

}

}

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(0)); //по-настоящему случайная генерация массива

cout << "Введите размер матрицы: ";

cin >> n;

cout << "Матрица №1: \n";

vis = (int\*)calloc(n, 3);

p = (int\*\*)calloc(n, 3);

SpecMatrix = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SpecMatrixRezerve = (SmegnList\*)calloc(n, sizeof(SmegnList\*));

SpecMatrixRezerve = CreateList();

for (int i = 0; i < n; i++) {

p[i] = (int\*)calloc(n, 3);

vis[i] = 0;

SpecMatrix[i] = CreateList(); //Создаём список смежности

AddSmegElem(SpecMatrix[i], i + 1); //Вставляем в начало каждого списка смежности номер отвечающей вершины

}

cout.precision(3 \* n);

//Задание 1.1//

for (int i = 0; i < n; i++) { //Генерация матрицы смежности

for (int m = 0+pogr; m < n; m++) {

p[i][m] = 0 + rand() % 2;

p[m][i] = p[i][m];

p[i][i] = 0;

}

pogr++;

}

for (int i = 0; i < n; i++) { //Создание списков смежности из матрицы смежности

for (int m = 0; m < n; m++) { if (p[i][m] == 1) { AddSmegElem(SpecMatrix[i], m + 1); } }

}

for (int i = 0; i < n; i++) { //Отрисовка матрицы смежности

for (int m = 0; m < n; m++) {cout << p[i][m] << " ";}

cout << endl;

}

cout << "\nСписки смежности для Мартицы №1:" << endl; //Отрисовка списков смежности

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix[i]); }

////

//Задание 1.2// Рекурсия - Поиск

cout << "\nВершины: " << endl;

DFS(0);

cout << "\n(Рекурсивная реализация)" << endl;

////

//Задание 1.3// Списки смежности - Поиск

DeepSearchSmeg();

cout << "(Поиск по спискам смежности)" << endl;

////

//Задание 2.1// Не рекурсия(цикл) - Поиск

NoRecSearch();

cout << "\n(Не рекурсивная реализация)" << endl;

////

return 0;

}

Пояснительный текст к программе:

Данная программа осуществляет обход графа в глубину. Работа алгоритма продолжается до тех пор, пока существуют непосещённые вершины.

Так же данная программа осуществляет процедуру обхода в глубину для графа, представленного списками смежности. А для матричной формы представления графов выполняет преобразование рекурсивной реализации обхода графа к не рекурсивной.

Результаты работы программы:

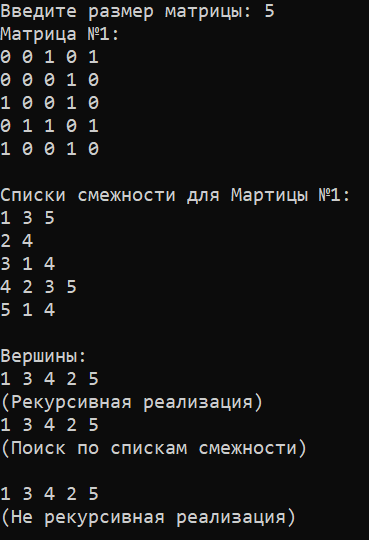


Рисунок №1 – Результат работы программы.

Вывод: При выполнении данной лабораторной работы мы смогли научиться осуществлять обход графа в глубину.